Ruby - Feature #17608

Compact and sum in one step

02/04/2021 06:18 AM - sawa (Tsuyoshi Sawada)

Status:	Rejected
Priority:	Normal
Assignee:	
Target version:	

Description

Many use cases of Array#sum are preceded with the compact method or are followed by a block to ensure the value is addable.

```
a = [1, nil, 2, 3]

a.sum # !> TypeError

a.compact.sum # => 6

a.sum{_1 || 0} # => 6
```

I propose there should be a way to do that in one step. I request either of the following:

A. Change the current behaviour to skip nils.

```
a.sum # => 6
```

B. Array#filter_sum method

```
a.filter_sum # => 6
```

C. An option for Array#sum

```
a.sum(compact: true) # => 6
```

History

#1 - 02/04/2021 06:19 AM - sawa (Tsuyoshi Sawada)

- Description updated

#2 - 02/04/2021 06:19 AM - sawa (Tsuyoshi Sawada)

- Description updated

#3 - 02/04/2021 06:22 AM - sawa (Tsuyoshi Sawada)

- Description updated

#4 - 02/04/2021 06:24 AM - sawa (Tsuyoshi Sawada)

- Description updated

#5 - 02/04/2021 07:17 PM - Eregon (Benoit Daloze)

a.sum{_1 || 0} seems more than good enough.

I don't think we want composite methods unless there is a significant performance advantage and it's more expressive.

#6 - 02/05/2021 01:27 PM - Hanmac (Hans Mackowiak)

you can use that nil.to_i returns 0

a.sum(&:to_i) #=> 6

#7 - 02/05/2021 02:25 PM - sawa (Tsuyoshi Sawada)

Hanmac (Hans Mackowiak) wrote in #note-6:

> [Y]ou can use [the fact] that nil.to_i returns 0
>
> a.sum(&:to_i) #=> 6

That won't work.

```
[1.2, nil, 3.6].sum{_1 || 0} # => 4.8
```

```
[1.2, nil, 3.6].sum(&:to_i) # => 4
```

#8 - 02/05/2021 04:17 PM - marcandre (Marc-Andre Lafortune)

Then use sum(&:to_f)...

I agree, we should close this request.

#9 - 02/05/2021 08:03 PM - Eregon (Benoit Daloze)

- Status changed from Open to Rejected

#10 - 02/05/2021 08:05 PM - Eregon (Benoit Daloze)

I closed it.
It seems clear there is no need for a new method or keyword argument, when a.sum{_1 || 0} already works well, is clear and concise.
In general it's a good idea to think about what to do with missing data anyway, ignoring might not always be appropriate.

#11 - 02/15/2021 05:48 AM - ko1 (Koichi Sasada)

> Many use cases of Array#sum are preceded with the compact

Can you count on your app or your observation?

```
ko1@aluminium:~$ gem-codesearch 'compact\.sum' | wc -l
75
```

not so many cases in gem-codesearch.

#12 - 02/15/2021 06:32 AM - sawa (Tsuyoshi Sawada)

ko1 (Koichi Sasada) wrote in #note-11:

> Can you count on your app or your observation?
>
> ```
> ko1@aluminium:~$ gem-codesearch 'compact\.sum' | wc -l
> 75
> ```
>
> not so many cases in gem-codesearch.

I searched on my company's private repository (the *** in the following was replaced by the repository name):

https://github.com/search?q=org%3A**********+compact.sum&type=Code

and got 94 results. The search itself may match not only exactly compact.sum but also strings like compact_sum, so I checked through manually, and all of them were compact.sum or a variant of it, like &.compact&.sum.

When I switch to all results on GitHub:

https://github.com/search?l=Ruby&q=compact.sum&type=Code

it says 2,134, but this time, it includes strings like compact_sum. I am not sure how many of them are genuine compact.sum, but I believe there are many.

You presented the result on gems, but my guess is that sum is a relatively newly introduced method, and so gems tend to avoid sum in the first place to make them compatible with old Rubies, hence, gems are biased as data source for this purpose.

Search on Stack Overflow returns 21 questions that include compact.sum.

https://stackoverflow.com/search?q=compact.sum

And please don't forget that, besides compact.sum, code fragments like sum{_1 || 0} are also relevant use cases.

#13 - 02/16/2021 07:13 AM - naruse (Yui NARUSE)

I think your code is actually rel.pluck(:col).compact.sum.
It can be written as rel.sum(:col)
https://api.rubyonrails.org/v6.1.0/classes/ActiveRecord/Calculations.html#method-i-sum

#14 - 02/16/2021 11:26 AM - mame (Yusuke Endoh)

I think this is an endless argument. An idiom ".compact.max" is 10x more frequent than ".compact.sum" in GitHub search. If we introduce Array#compact_max, then #compact_min and #compact_minmax should be also introduced. Next, why don't we need #compact_max_by, #compact_max_by, and #compact_minmax_by? Maybe some people want #compact_sort_by eventually.

We have never yet had combination methods #sort_uniq and #zip_map, which are clearly frequent idioms. You need to prove that .compact.sum is really special, e.g., it is a critical performance bottleneck in multiple real-world applications, it is indisputably frequent (like #filter_map), it is difficult to work around (like #flat_map), etc.

#15 - 03/16/2021 10:58 AM - Eregon (Benoit Daloze)

FWIW, this was confirmed as rejected by matz:
https://github.com/ruby/dev-meeting-log/blob/master/DevelopersMeeting20210216Japan.md#feature-17608-compact-and-sum-in-one-step-sawa